Vivado Design Suite Tutorial

Programming and Debugging

UG936 (v2012.4) December 18, 2012

XILINX®

Notice of Disclaimer
The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.

© Copyright 2012 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/25/12	2012.2	Initial Xilinx release.
10/16/12	2012.3	• Added section for Synplify EDIF flow. • Various editorial fixes and updated screen shots to track changes to dialog boxes.
12/18/2012	2012.4	Revalidated for the 14.3 release. Editorial updates only; no technical content updates.

XILINX®

Table of Contents

Revision History 2

Debugging with Vivado™ ILA 2.0 and Integrated Logic Analyzer

Introduction 4
Prerequisites 4
Objectives 4
Getting Started 5
Step 1: Creating and Implementing an RTL Project in the Vivado Integrated Design Environment 7
Step 2: Probing and Adding Debug IP 10
Step 3: Using the Vivado Logic Analyzer to Debug the Hardware 23

Appendix A: Additional Resources

Xilinx Resources 37
Solution Centers 37
References 37

XILINX

Debugging with Vivado™ ILA 2.0 and Integrated Logic Analyzer

Introduction

In this tutorial exercise, you will quickly learn how to debug your FPGA designs by inserting an Integrated Logic Analyzer (ILA) core using the Vivado™ Integrated Design Environment (IDE). You will take advantage of integrated Vivado logic analyzer functions to debug and discover some potential root causes of your design, thereby allowing you to quickly address issues.

Example RTL designs will be used to illustrate overall integration flows between Vivado logic analyzer, ILA 2.0, and Vivado IDE. In order to be successful using this tutorial, you should have some basic knowledge of Xilinx® ISE® Design Suite and Vivado Design Suite tool flows.

Prerequisites

A basic knowledge of Xilinx ISE Design Suite and Vivado Design Suite tool flows.

Objectives

This tutorial:

- Shows you how to take advantage of integrated Vivado logic analyzer features in the Vivado design environment that make the debug process faster and simpler.
- Provides specifics on how to use the Vivado Integrated Design Environment (IDE) and the Vivado logic analyzer to debug some common problems in FPGA logic designs.

After completing this tutorial, you will be able to:

- Validate and debug your design using the Vivado Integrated Design Environment and the Integrated Logic Analyzer (ILA) core.

- Understand how to create an RTL project, probe your design, insert an ILA 2.0 core, and implement the design in the Vivado Integrated Design Environment.
- Generate and customize an IP core netlist in the Vivado Integrated Design Environment.
- Debug the design using Vivado logic analyzer in real-time, and iterate the design using the Vivado Integrated Design Environment and a KC705 Evaluation Kit Base Board that incorporates a Kintex™-7 device.

Getting Started

Setup Requirements

Before you start this tutorial, make sure you have and understand the hardware and software components needed to perform the steps. The following subsections list the requirements.

Software

Vivado Design Suite 2012.3

Hardware

Kintex-7 FPGA KC705 Evaluation Kit Base Board.

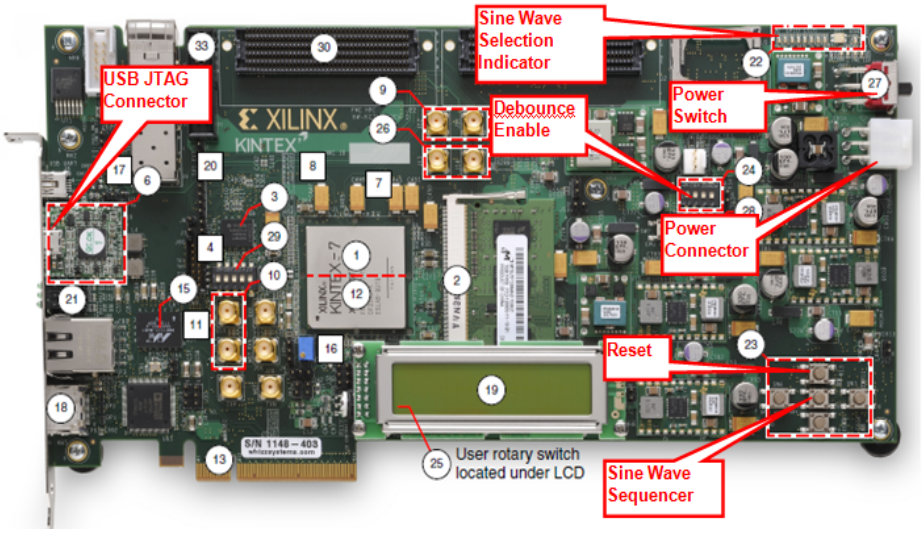

Figure 1: **KC705 Board Showing Key Components**

Tutorial Design Components

The design includes:

- A simple control state machine.
- Three sine wave generators using AXI-Streaming interface, native DDS Compiler.
- Common push buttons (`GPIO_BUTTON`).
- DIP switches (`GPIO_SWITCH`).
- LED displays (`GPIO_LED`).

Push Button Switches: Serve as inputs to the debounce and control state machine circuits. Pushing a button generates a high-to-low transition pulse. Each generated output pulse is then used as an input into the state machine.

DIP Switch: Enables or disables a debounce circuit.

Debounce Circuit: In this example, when enabled, provides a clean pulse or transition from high to low. Eliminates a series of spikes or glitches when a button is pressed and released.

Sine Wave Sequencer State Machine: Captures and decodes input pulses from the two push button switches. Provides sine wave selection and indicator circuits, sequencing between 00, 01, 10, and 11 (zero to three).

LED Displays: `GPIO_LED_0` and `GPIO_LED_1` display selection status from the state machine outputs, each of which represents a different sine wave frequency: high, medium, and low.

Tutorial design files: For details on locating design files, see Getting Started, page 7.

Board Support and Pinout Information

Table 1-1: **: Pinout Information for the KC705 Board**

	Pinout Locations	Function
CLK_N	AD11	Clock
CLK_P	AD12	Clock
GPIO_BUTTONS[0]	AA12	Reset
GPIO_BUTTONS[1]	AG5	Sine Wave Sequencer
GPIO_SWITCH	Y28	Debounce Circuit Selector
LEDS_n[0]	AB8	Sine Wave Selection[0]
LEDS_n[1]	AA8	Sine Wave Selection[1]
LEDS_n[2]	AC9	Reserved
LEDS_n[3]	AB9	Reserved

Step 1: Creating and Implementing an RTL Project in the Vivado Integrated Design Environment

To create and implement an RTL project, do the following:

- Get started by unzipping the tutorial source files and launching the Vivado Integrated Design Environment (IDE).
- Create a New Project with the New Project Wizard.
- Synthesize the design.

Getting Started

1. In your `C:` drive, create a folder called `/Vivado_Debug`.
2. Find the tutorial source files.

   Design file location:
   http://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_3/ug936-vivado-debug-design-files.zip

   The tutorial and design files might be updated or modified in between software releases on the Xilinx website, where you can download the latest version of the materials.

3. Unzip the tutorial source file to the `/Vivado_Debug` folder.
4. When unzipped, look in `Vivado_Debug/src` for the files and folder shown in Figure 2.

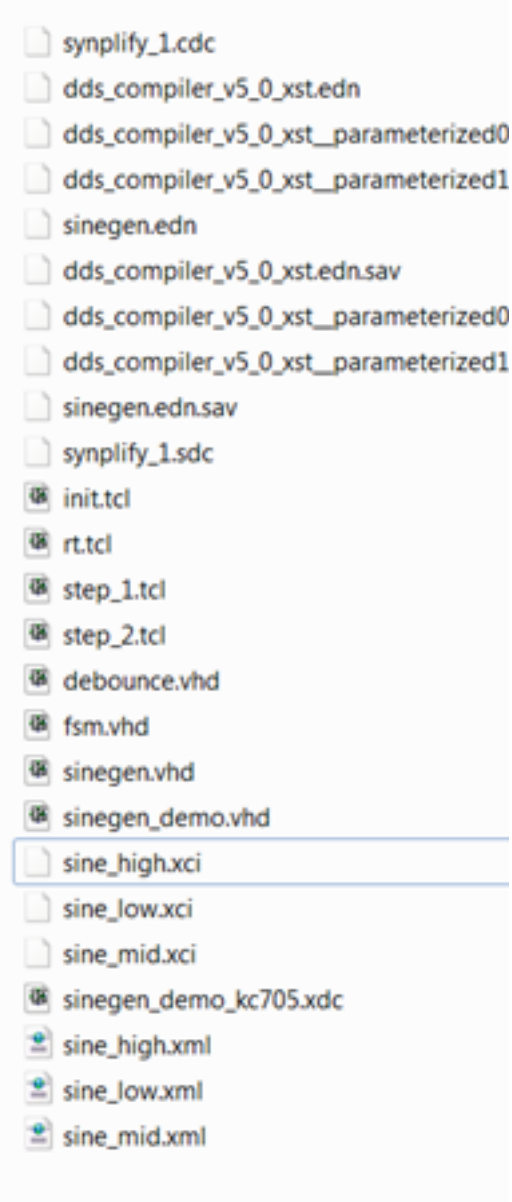

Figure 2: **Tutorial Design File Set**

Creating a Project with the Vivado New Project Wizard

To create a project, use the New Project wizard to name the project, to add RTL source files and constraints, and to specify the target device.

IMPORTANT: *If you are already familiar with creating a new project in Vivado, use the script provided in step_1.tcl to perform the tasks described in Step 1. Make sure you cd to the same level of the ./src folder before sourcing the step_1.tcl, i.e.* `cd c:/Vivado_Debug/src`

1. Invoke the Vivado IDE.
2. In the Getting Started screen, click **Create New Project** to start the New Project wizard.
3. Select **Next** to continue to the next screen.
4. In the **Project Name** screen, name the new project proj_step1 and provide the project location (`C:\Vivado_Debug`). Ensure that **Create Project Subdirectory** is checked and click **Next**.
5. In the **Project Type** screen, specify the **Type of Project** to create as **RTL Project** and click **Next**.
6. In the **Add Sources** screen:
   a. Set **Target Language** to **VHDL**.
   b. Click the **Add Files** button.
   c. In the **Add Source Files** dialog box, navigate to the `/src directory`.
   d. Select all VHD source files, and click **OK**.
   e. Verify that the files are added, and **Copy Sources into Project** is checked. Click **Next**.
7. In the **Add Existing IP (optional)** dialog box:
   a. Click the **Add Files** button.
   b. In the **Add Configurable IP** dialog box, navigate to the `/src directory`.
   c. Select all XCI source files, and click **OK**.
   d. Verify that the files are added, and **Copy Sources into Project** is checked. Click **Next**.
8. In the **Add Constraints (optional)** dialog box, the provided XDC file should automatically appear in the main window. Another file called synplify_1.sdc should also appear as a part of the included files. Select and remove the synplify_1.sdc file by clicking the **Remove Selected File** button.

   If none of the files appear in the dialog box by default, click the **Add Files** button, navigate to the `/src` directory and select `sinegen_demo_kc.xdc`.

   Click **Next** to continue.

   Note: The `synplify_1.sdc` file is needed for the Synopsys Synplify Pro flow.
9. In the **Default Part** dialog box, specify the **xc7k325tffg900-2** part for the KC705 platform. It is easiest to specify **Boards** for this target device and select **Kintex-7 KC705 Evaluation Platform**, and click **Next.**
10. Review the **New Project Summary** screen. Verify that the data appears as expected, per the steps above, click **Finish**.

Note: It might take a moment for the project to initialize.

After you exit the New Project wizard, use the Project Manager in the Vivado IDE main window to add IP and to synthesize the design.

Synthesizing the Design

1. In the Project Manager, click **Project Settings**, change **Synthesis -flatten_hierarchy** option to **none**, and click **OK.**

Note: The reason for changing this setting to **none** is to prevent the synthesis tool from performing any boundary optimizations for this tutorial.

2. In the left panel, expand the **Synthesis** folder, and click the **Run Synthesis** button.

Note: When synthesis runs, a progress indicator appears, showing that synthesis is occurring. This could take a few minutes.

3. In the **Synthesis Completed** dialog box, click **Cancel**. You will implement the design later.
4. Select **File > Save Project As** and save the project as proj_step2. Saving the project to a new file allows you to more easily resume the tutorial if you do not wish to complete all the steps in one sitting.

Step 2: Probing and Adding Debug IP

Using the Netlist Insertion Method

To add a Vivado ILA 2.0 core to the design, take advantage of the integrated flows between the Vivado IDE and Vivado logic analyzer.

IMPORTANT: *If you are already familiar with adding Debug Nets and adding Debug IP using the Core Insertion method, use the provided* `step_2.tcl` *to perform the same tasks as described below. Make sure you cd into the same level of the* ***`./src`*** *folder before sourcing the* `step_2.tcl`*, i.e.* `cd c:/Vivado_Debug/src`

You will accomplish the following tasks:

- Add debug nets to the project.
- Run the Set Up Debug Wizard.
- Implement and open the design.
- Generate the Bitstream.

Adding Debug Nets to the Project

Working in the **project_step2** project:

Following are some examples of how to add debug nets using the Vivado IDE:

- Add mark_debug attribute to the target XDC file

  ```
  set_property mark_debug true [get_nets sine*]
  ```

  Note: Use these attributes in synthesized designs only. Do not use then with pre-synthesis or elaborated design netlists.

- Add mark_debug attribute to HDL files

  VHDL

  ```
  attribute mark_debug : string;
  attribute keep : string;
  attribute mark_debug of sine     : signal is "true";
  attribute mark_debug of sineSel  : signal is "true";
  ```

  Verilog

  ```
  (* mark_debug = "true" *) wire sine;
  (* mark_debug = "true" *) wire sineSel;
  ```

- Right-click and select **Mark Debug** or **Unmark Debug** on **Synthesis** netlist.
- Use a Tcl prompt to set the mark_debug attribute. For example, `set mark_debug true [get_nets sine*]`. This applies the mark_debug on the current, open netlist.

In this tutorial, you will learn how to add debug nets to HDL files and the synthesized design using Vivado IDE.

1. From the Synthesis pull-down, click **Open Synthesized Design**.

   Note: Before proceeding, make sure that the **Flow Navigator** on the left panel is enabled. Use **Ctrl-Q** to toggle off and on. Secondly, the window layout must be set to **Debug.**

   At this point you will get a Synthesis is Out-of-Date warning. Click **Open Design**.

2. Click the **Debug** tab if it is not already selected.
3. Expand **Unassigned Debug Nets** folder. In Figure 4 you should see those debug nets that were tagged in the `sinegen_demo.vhd` with mark_debug attributes, as shown in Figure 3.

```
attribute mark_debug : string;
attribute mark_debug of GPIO_BUTTONS_db   : signal is "true";
attribute mark_debug of GPIO_BUTTONS_dly  : signal is "true";
attribute mark_debug of GPIO_BUTTONS_re   : signal is "true";
component sinegen
  port
  (
    clk   : in    std_logic;
    reset : in    std_logic;
    sel   : in    std_logic_vector(1 downto 0);
    sine  : out   std_logic_vector(19 downto 0)
  );
end component;
```

Figure 3: **VHDL Example Using MARK_DEBUG Attributes**

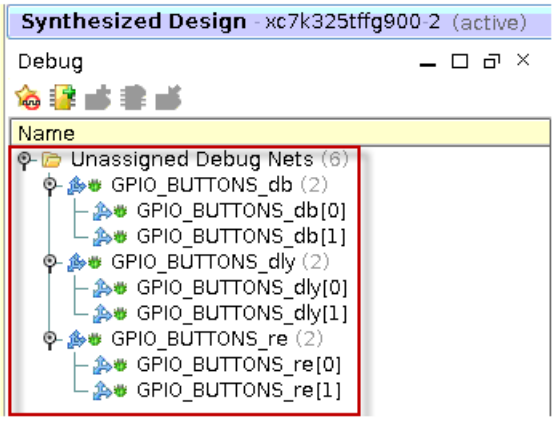

Figure 4: **Unassigned Debug Nets Post-synthesis**

4. Select the **Netlist** tab to expand **Nets**. Select the following nets as shown in Figure 5 for debugging.
   - `GPIO_BUTTONS_IBUF(2)` - Nets folder under the top-level hierarchy
   - `sine(20)` - Nets folder under the `U_SINEGEN` hierarchy
   - `sel(2)` -Nets folder under the `U_SINEGEN` hierarchy

   Note: These signals represent the significant behavior of this design and will be used to verify and debug the design in subsequent steps.
5. Right-click the selected nets and select **Mark Debug.**

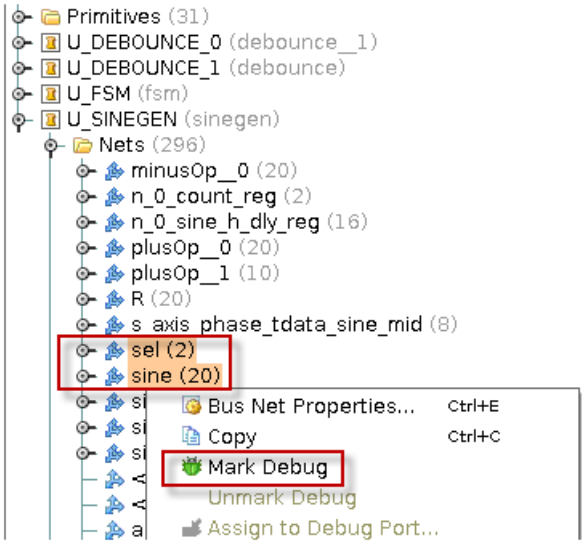

Figure 5: **Adding Nets from the Netlist Tab**

Note: With the **Debug** tab selected, you can see the unassigned nets you just selected. In Vivado IDE, you can also see the green bug icon next to each scalar or bus, which indicates that a net has the attribute `mark_debug = true` as shown in Figure 6.

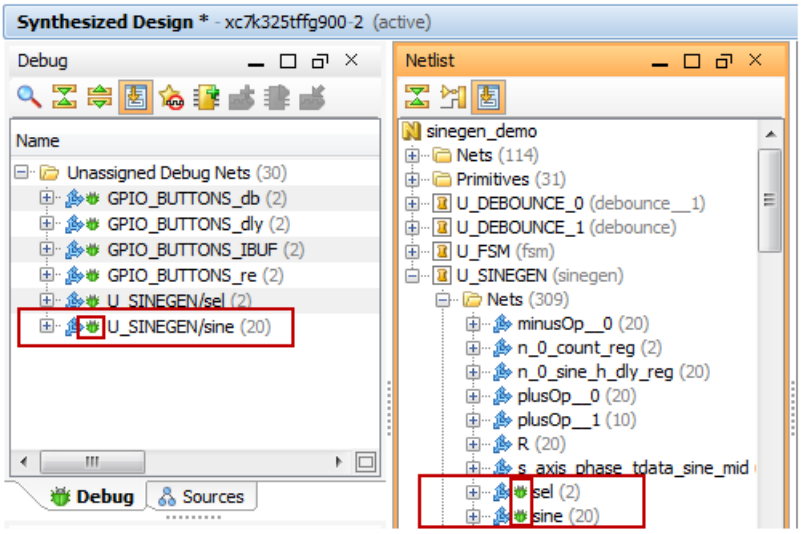

Figure 6: **Mark_Debug Nets Post-synthesis**

Running the Set Up Debug Wizard

1. From the **Debug** tab (by clicking on the tab) or **Tools** menu, select **Set Up Debug**. The **Set Up Debug** wizard opens.
2. Click through the wizard to create Vivado logic analyzer debug cores, keeping the default settings.

Using the HDL Instantiation Method

The HDL Instantiation method is one of the two methods that are supported in Vivado Debug Probing. For this flow, a you will have to generate an ILA 2.0 IP using the Vivado IP Catalog and instantiate the core in a design manually as you would with any other IP.

If you choose to use this method then you can skip all the instructions described under the **Using the Netlist Insertion Method** section.

1. With this flow, use `sinegen_demo_inst.vhd` as the top-level VHDL file instead of `sinegen_demo.vhd`.
2. Add `ila_v2_0_0.xci` IP.
3. Synthesize the design.

HDL Debug Flow Using Synopsys Synplify Pro Synthesis Tool

Skip this section in its entirety if you are not using the Synopsys Synplify Pro synthesis tool as a part of your design flow.

This simple tutorial shows how to do the following:

- Create a Synplify Pro project for the wave generator design.
- Mark nets for debug in the Synplify Pro constraints file as well as VHDL source files.
- Synthesize the Synplify Pro project to create an EDIF netlist.
- Create a Vivado project based on the Synplify Pro netlist.

Use the Vivado IDE to setup and debug the design from the synthesized design using Synplify Pro.

Create a Synplify Pro Project

1. Launch Synplify Pro and select **File > New**. Set **File Type** to **Project File (Project)** as highlighted in Figure 7. In the **New File Name** box, enter **synplify_1**. Click **OK**.

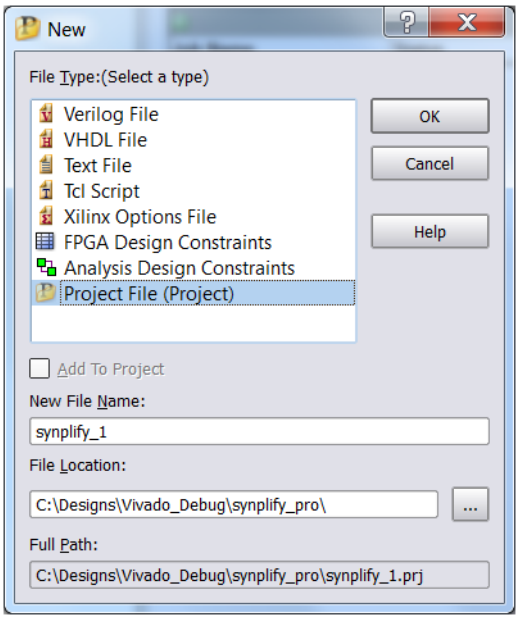

Figure 7: **Synplify Pro New Project Dialog Box**

2. In the left panel of the Synplify Pro window, click **Add File** as shown in Figure 8.

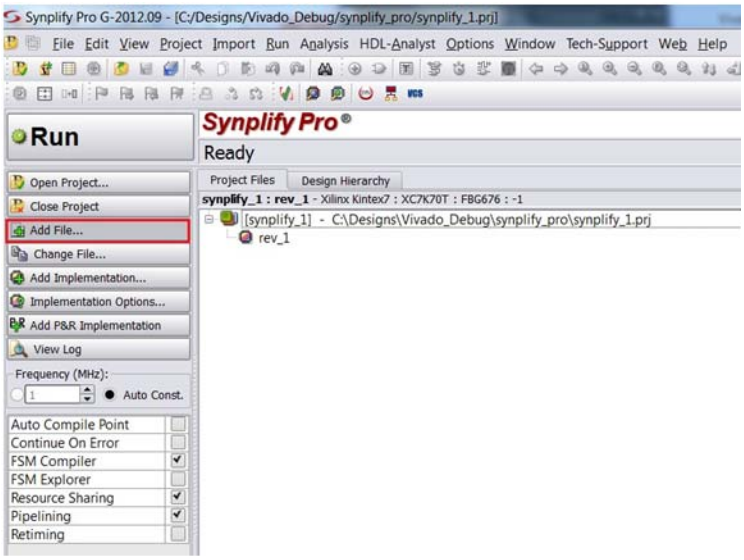

Figure 8: **Adding Files to a Synplify Pro Project**

This will bring up the **Add Files to Project** dialog box as shown in Figure 9.

a. First, change the **Files of Type** to **HDL File**. This shows all the VHDL source files in the `Vivado_Debug\src` folder. Select the following three files by pressing the **Ctrl** key and clicking on them.

- `debounce.vhd`
- `fsm.vhd`
- `sinegen_demo.vhd`

Do not add the fourth file, `sinegen.vhd`, this Synplify Pro project. That block is treated as a black box. Later, in Vivado IDE, you will bring in the synthesized design for this black box.

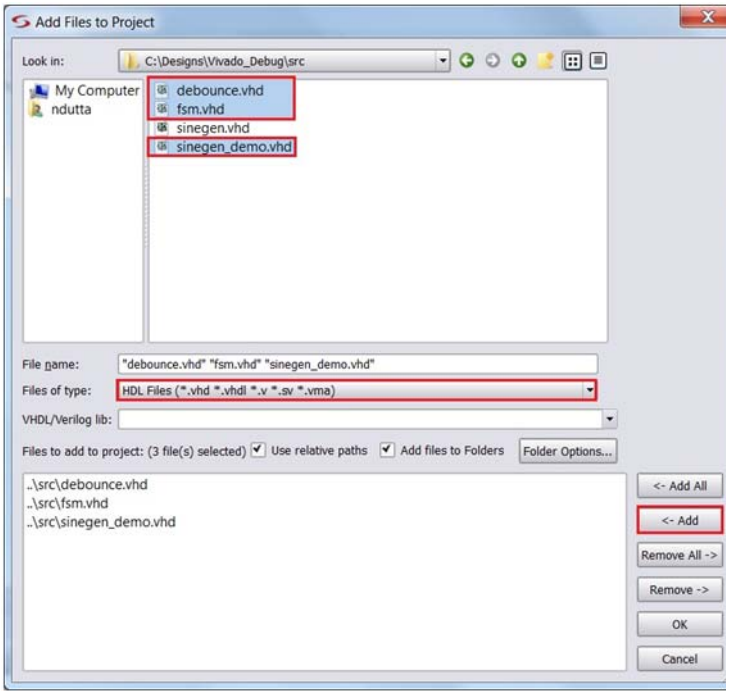

Figure 9: **Adding VHDL Source Files to the Synplify Pro Project**

b. In the same dialog box set **Files of type** to **Constraints File**. This shows the `synplify_1.sdc` file. Select the file and click **Add** as shown in Figure 10. Click **OK**.

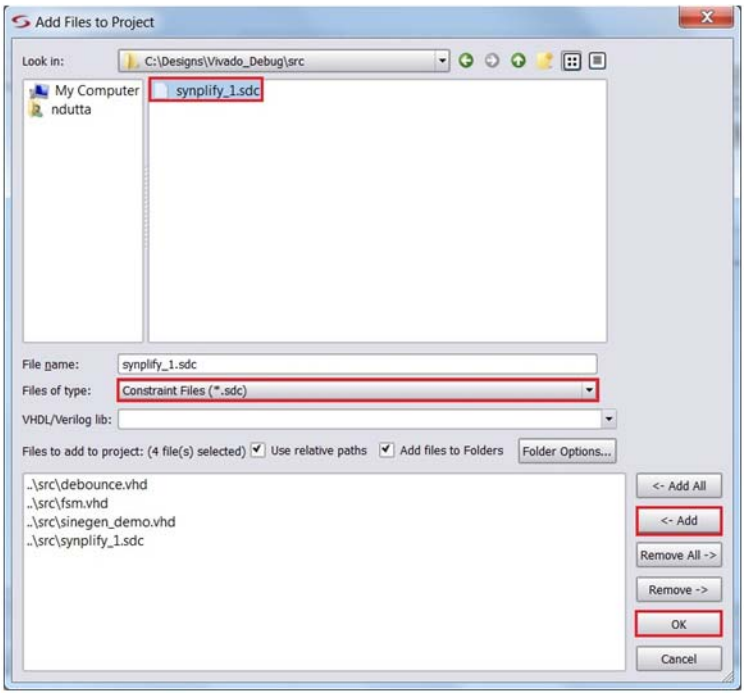

Figure 10: **Adding SDC Constraints File to the Synplify Pro Project**

c. In the same dialog box set **Files of type** to **Compiler Directives File**. This shows the `synplify_1.cdc` file. Select the file and click **Add** as shown in Figure 11. Click **OK**.

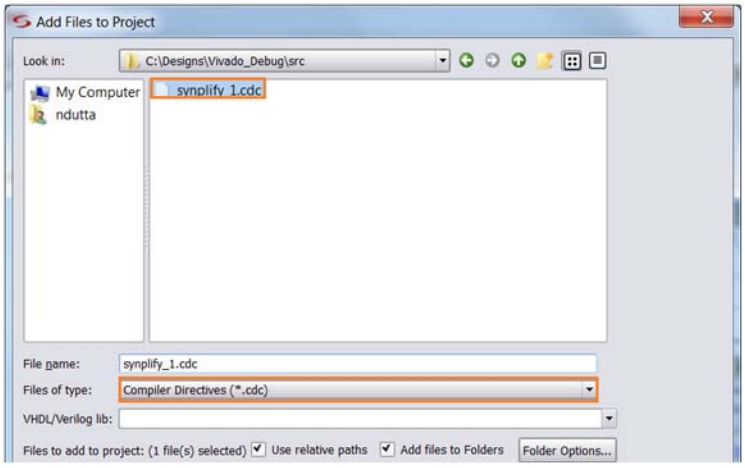

Figure 11: **Adding CDC Constraints File to the Synplify Pro Project**

3. Now, you need to set the implementation options. Click **Implementation Options** in the Synplify Pro window as shown in Figure 12.

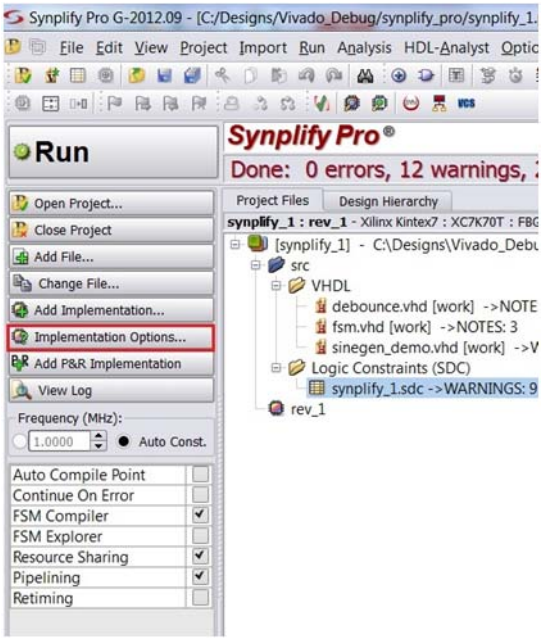

Figure 12: **Opening Implementation Options in Synplify Pro**

4. This brings up the **Implementation Options** dialog box as shown in Figure 13. In the **Device** tab, set **Technology** to **Xilinx Kintex7**, **Part** to **XC7K325T**, **Package** to **FFG900** and **Speed** to **-2**. Leave all the other options at their default values. Click **OK**.

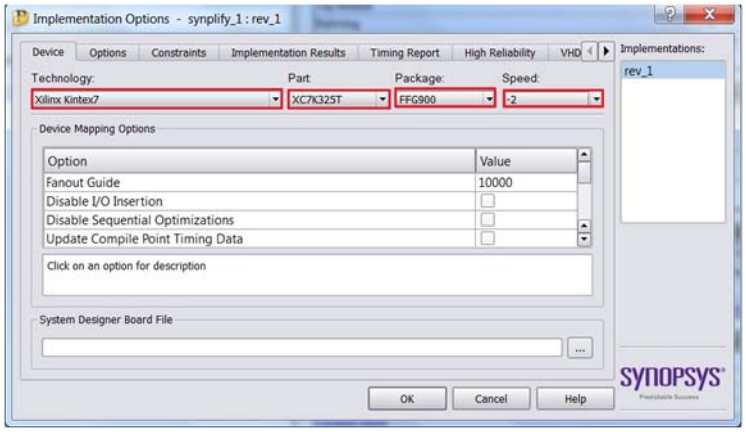

Figure 13: **Specifying Implementation Options in Synplify Pro**

Mark Nets for Debug in VHDL

Use the mark_debug attribute in the VHDL source files as described in Adding Debug Nets to the Project, page 11. These attributes are already placed in the `sinegen_demo.vhd`, VHDL source files of this tutorial. Open the `sinegen_demo.vhd` file. Uncomment lines

69-72 that specify the attributes for marking debug nets in Synplify Pro as shown in Figure 14.

```
68   -- Attributes for Synplify Pro
69   -- attribute syn_keep : boolean;
70   -- attribute syn_keep of GPIO_BUTTONS_db   : signal is true;
71   -- attribute syn_keep of GPIO_BUTTONS_dly  : signal is true;
72   -- attribute syn_keep of GPIO_BUTTONS_re   : signal is true;
73
```

Figure 14: **Specifying Implementation Options in Synplify Pro**

Mark Nets for Debug in CDC

The `synplify_1.sdc` file contains various kinds of constraints such as pin location, I/O standard, and clock definition. The `synplify_1.cdc` file contains directives for the compiler. Here is where the nets of interest to us, that are marked for debug, are located. The attribute and the nets selected for debug are shown in Figure 15.

```
#
# Attributes that are needed to mark_debug the nets that are needed to be viewed in ILA
#
define_attribute  -comment {Mark u_sinegen as black box} {v:work.sinegen} {syn_black_box} {1}
define_attribute  -comment {Set no_prune on sinegen} {v:work.sinegen} {syn_no_prune} {1}
define_attribute  -comment {Mark entire bus for debug} {n:sine[*]} {mark_debug} {"true"}
define_attribute  -comment {Mark entire bus for debug} {n:sinegen.sel[*]} {mark_debug} {"true"}
```

Figure 15: **Synplify Pro Constraints in CDC Files**

In the above constraints, `sinegen` has been defined as a black box by using the syn_black_box attribute. Second, the syn_no_prune attribute has been used so that the I/Os of this block are not optimized away. Finally, two nets, `sine[20:0]` and `sel[1:0]` have been assigned the mark_debug attribute such that these two nets should show up in the synthesized design in Vivado IDE for further debugging. For further information on these attributes, please refer to the *Synplify Pro User Manual* and *Synplify Pro Reference Manual*.

Synthesize the Synplify Project

1. Before implementing the project, you need to set the name for the output netlist file. By default, the name of the output netlist file is `synplify_1.edf`. To change the name of the output file, type the following command at the Tcl command prompt:

   ```
   %project -result_file "./rev_1/sinegen_demo.edf"
   ```

   You will use this file in Vivado IDE.

2. With all the project settings in place, click the **Run** Button in the left panel of the **Synplify Pro** window to start synthesizing the design.

3. During synthesis, status messages will appear in the **Tcl Script** tab. Warning messages are expected, but there should not be any Error messages. To see detailed messages, click the **Messages** tab in the bottom left-hand corner of the **Synplify Pro** console.
4. When synthesis completes, the output netlist is written to the file: `rev_1/sinegen_demo.edf`.
5. [Optional] To view the netlist select **View > View Result File**. The mark_debug attributes can be seen in this netlist.
6. Click **File > Save All** to save the project, then click **File > Exit**.

Create EDIF Netlists for the Black Box Created in Synplify Pro

The black box sinegen created in the Synplify Pro project, contains the Direct Digital Synthesizer IP. You need to create a synthesized design for this block. To do this, create an RTL type project in Vivado IDE by following the steps outlined below.

1. Launch Vivado IDE and select **Create New RTL Project**. This opens up the **New Project** wizard. Click **Next**.
2. Under **Project Name,** set the project name to **proj_step3**. Click **Next**.
3. Under **Project Type,** select **RTL Project**. Click **Next**.
4. Under **Add Sources**, click **Add Files**, navigate to the `Vivado_Debug/src` folder and select the `sinegen.vhd` file. Set **Target Language** to **VHDL**. Ensure that **Copy sources into project** box is checked. Click **Next**.
5. Under **Add Existing IP**, click **Add Files**, navigate to the `Vivado_Debug/src` folder and select the `sine_high.xci`, `sine_low.xci`, and `sine_mid.xci` files. Click **Next**.
6. Under **Add Constraints**, two `.sdc` files are automatically added to the project. These files are not needed for this step. Remove them from this project by clicking the **Remove Selected File** button on the right of the dialog box. Click **Next**.
7. Under **Default Part**, select **Boards** and then select the **Kintex-7 KC705 Evaluation Platform**. Click **Next**.
8. Under **New Project Summary**, ensure that all the settings are correct and click **Finish**.
9. Once the project has been created, in Vivado Flow Navigator, under the Project Manager folder, click **Project Settings**. In the pop-up dialog box, in the left panel, click **Synthesis**. From the pull down menu on the right panel, set **–flatten_hierarchy** to **none** , and check the **–no_iobuf** box.
10. In Vivado IDE Project Manager, under **Synthesis Folder**, click on **Run Synthesis**. When synthesis completes the **Synthesis Completed** dialog box appears. Select **Open Synthesized Design** and click **OK**.
11. Now you need to write the netlist file for all the components used in the singen block. The four netlist files used in this tutorial are already provided as a part of the source

files. However, you can overwrite them by using your own netlist files. To do this use the following Tcl command in the Tcl console of Vivado IDE.

```
write_edif  -force ../Vivado_Debug/src/sinegen.edn
```

Ensure that the path specified to the src folder is correct. At this point, you should see four `.edn` files in the `Vivado_Debug/src` folder as shown below:

- `dds_compiler_v5_0_xst.edn`
- `dds_compiler_v5_0_xst__parameterized0.edn`
- `dds_compiler_v5_0_xst__parameterized1.edn`
- `sinegen.edn`

These files are provided in the `Vivado_Debug/src` folder. You may choose to overwrite these files using the `write edif -force` command option as shown above.

12. Click **File > Exit** in Vivado IDE.

Create a Post Synthesis Project in Vivado IDE

1. Launch Vivado IDE and select **Create New Project**. This opens up the **New Project** wizard. Click **Next**.
2. Set the **Project Name** to **proj_step4**. Click **Next**.
3. Under **Project Type**, select **Post-synthesis Project**. Click **Next**.
4. Under **Add Netlist Sources**, click **Add Files**, navigate to the `Vivado_Debug/synplify_pro/rev_1` folder, and select **sinegen_demo.edf**.
5. Add the four netlist files created in the previous section. Click on **Add Files** again, navigate to the `Vivado_Debug/src folder` and select the following files:
   - `sinegen.edn, dds_compiler_v5_0_xst.edn`
   - `dds_compiler_v5_0_xst_parameterized0.edn,`
   - `dds_compiler_v5_0_xst_parameterized1.edn.`

   Enure that **Copy Sources into Project** is checked. Click **Next**.
6. Under **Add Constraints**, some `.xdc` files should be automatically populated. Remove these files by selecting them and clicking the **Remove Selected File** button on the right of the dialog box. Click **Add Files** and navigate to the `Vivado_debug/src` folder and select the **sinegen_demo_kc705.xdc** file. This file has the appropriate constraints needed for this Vivado project. Ensure that **Copy Constraints into Project** is checked. Click **Next**.
7. Under **Default Part**, select **Boards** and then select the **Kintex-7 KC705 Evaluation Platform**. Click **Next**.

8. Under **New Project Summary**, ensure that all the settings are correct and click **Finish**.

Add Debug Nets to the Project

1. In Vivado IDE, in Project Manager in the left panel, select **Open Synthesized Design** from the **Synthesis** folder.
2. You should be able to see all the nets that are marked for debug as shown in Figure 16.

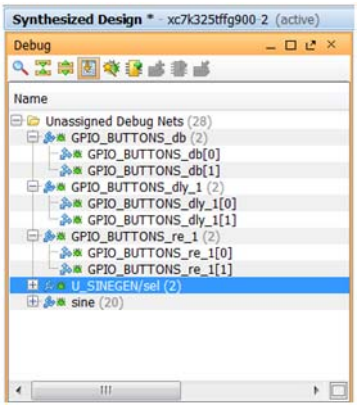

Figure 16: **Nets added for debug through the Synplify Pro Flow in Vivado IDE**

Running the Set up Debug Wizard

1. Right-click the **Debug** tab or select the **Tools** menu, select **Set up Debug**. The **Set up Debug** wizard opens.
2. Click through the wizard to create Vivado logic analyzer debug cores, keeping the default settings.

Implementing the Design and Generating the Bitstream

1. In the left panel, main window of the Vivado IDE, click the **Run Implementation** button.
2. In the **Save Project** pop-up menu, click **Save**.
3. When the implementation process ends, an **Implementation Completed** dialog box opens.

   Note: Implementation could take about 10 minutes.
4. In the **Implementation Completed** dialog box, click **Cancel**.
5. In the upper right-hand corner, click on **more info** and then **Force up-to-date** as shown in Figure 17. This step will prevent the tools from re-synthesizing and re-implementing the design.

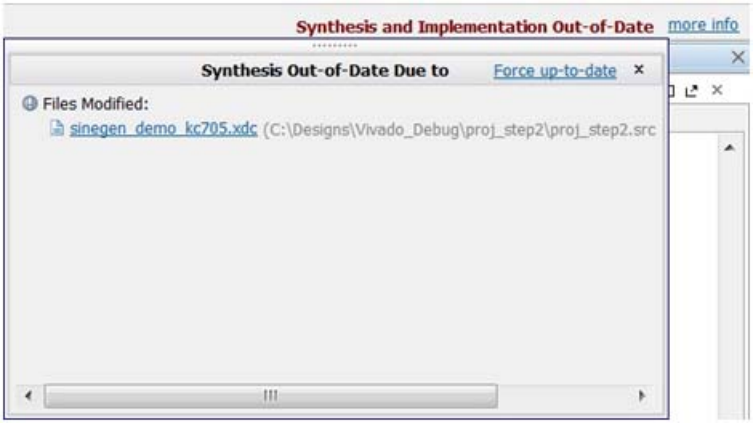

Figure 17: **Synthesis Out-of-Date**

6. Expand the **Program and Debug** folder, and click **Generate Bitstream**. Click **OK** when the **Bitstream Generation Completed** pop-up appears to let you know the process is finished.

Step 3: Using the Vivado Logic Analyzer to Debug the Hardware

Now that you have added an ILA 2.0 to your design and connected it to your debug nets, implemented your design, and created a bitstream file, you are almost ready to use the integrated Vivado logic analyzer feature to interact with the ILA core. However, before you do so, you need to make sure you have KC705 hardware plugged into a machine and are running a cse_server application on that machine.

In this step, you learn:

- How to debug the design using the Vivado logic analyzer.
- How to use the current supported Tcl commands to communicate with your target board (KC705).
- How to discover and correct a circuit problem by identifying unintended behaviors of the push button switch.
- Some useful techniques for triggering and capturing design data.

Verifying Operation of the Sine Wave Generator

After doing some setup work, you will use Vivado logic analyzer to verify that the sine wave generator is working correctly. The two primary objectives are to verify that:

- All sine wave selections are correct.
- The selection logic works correctly.

Setting Up

If you plan to connect remotely, you will need to make sure you have KC705 hardware plugged into a machine and are running a cse_server application on that machine. If you plan to connect locally, skip step 1-4 below.

1. Connect the Digilent USB JTAG cable of your KC705 board to a Windows machine's USB port.
2. Ensure that the board is plugged in and powered on.
3. Assuming you are connecting your KC705 board to a 64-bit Windows machine and you will be running the cse_server from the network instead of your local drive, open a cmd prompt and type the following:

   ```
   <Xilinx_Install>\ISE_DS\ISE\bin\nt64\cse_server
   ```

4. Leave this cmd prompt open while the cse_server is running. Note the machine name that you are using, this will be used later when opening a connection to this instance of the cse_server application.

If you plan to connect locally, ensure that you have your KC705 hardware plugged into a Windows machine and then perform the following steps:

1. Connect the Digilent USB JTAG cable of your KC705 board to a Windows machine's USB port.
2. Ensure that the board is plugged in and powered on.
3. Turn DIP switch positions on SW8 (Debounce Enable) to the OFF position.

You are now ready to use the Vivado logic analyzer.

1. In the Vivado IDE, from the **Program and Debug** drop-down list, select **Open Hardware Session**.

Note: Note that the window layout changed to look like the Figure 18:

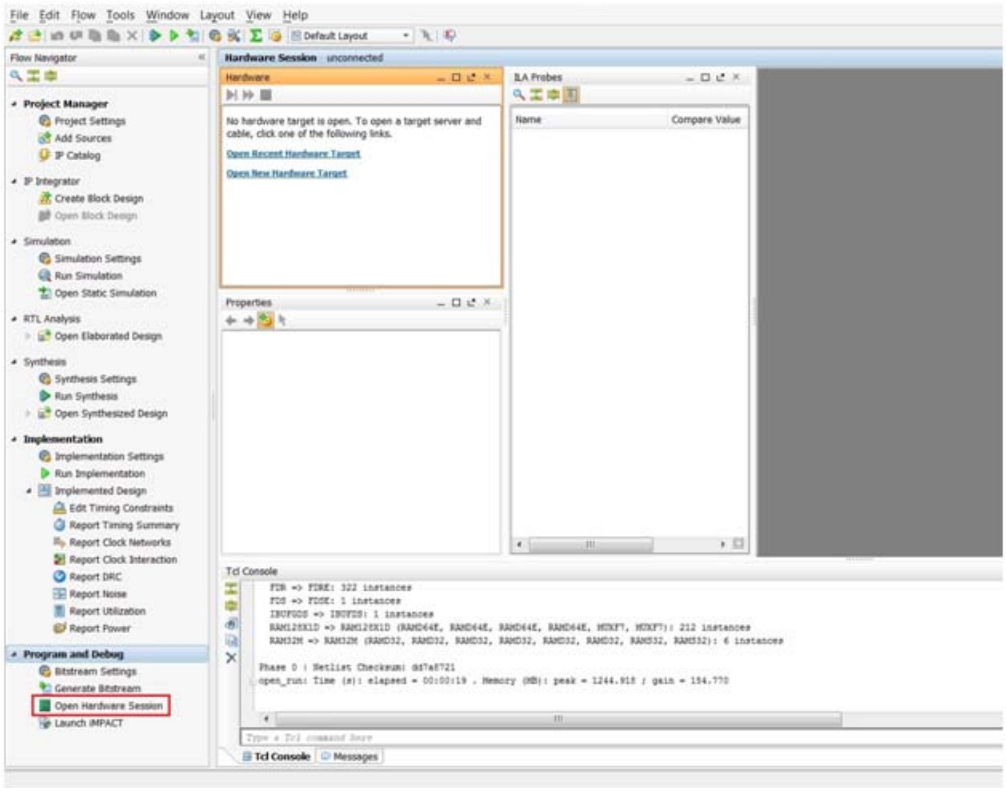

Figure 18: **Window Layout for Program and Debug**

2. Click on the **Open New Target** link in the **Hardware** view and select **Open Hardware Target Wizard...** .
3. Click **NEXT** in the wizard.
4. Type the name of the server in the text field and click **Next.**

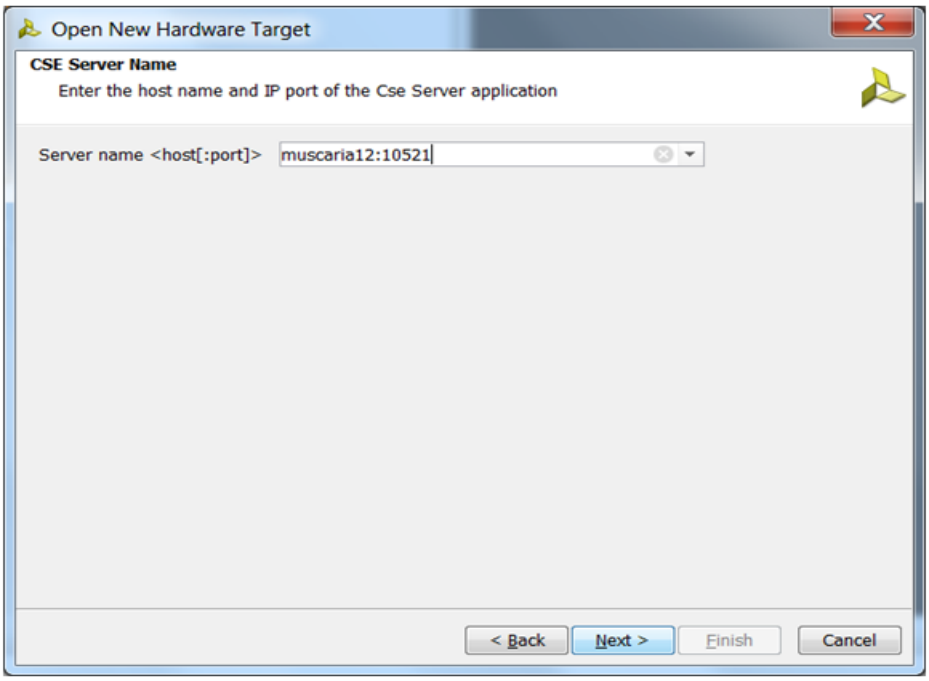

Figure 19: **CSE Server Name and Server Status**

Note: Depending on your connection speed, this may take about 10~15 seconds

5. If there is more than one target connected to the cse_server you will see multiple entries in the **Select Hardware Target** dialog box. In this tutorial, there is only one target as shown in Figure 20. Select the desired target and click **Next**

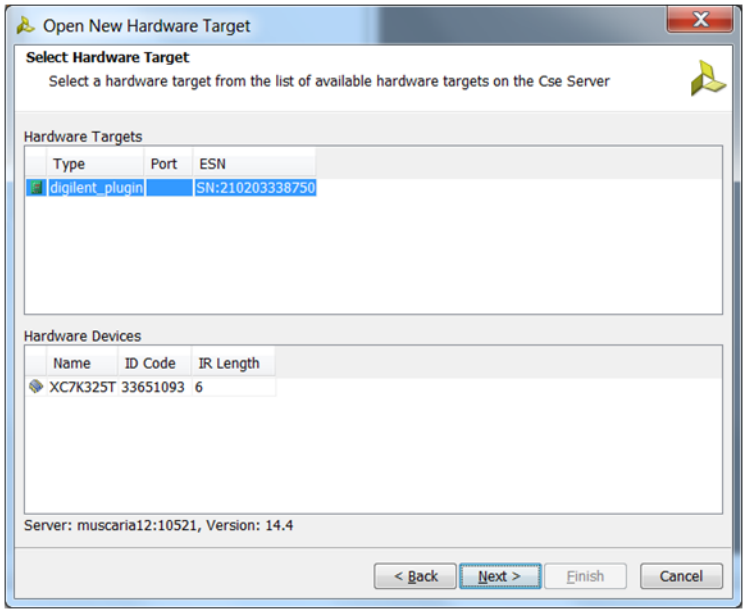

Figure 20: **CSE Hardware Target Chooser**

6. Leave these settings at their default values as shown in Figure 21. Click **Next**.

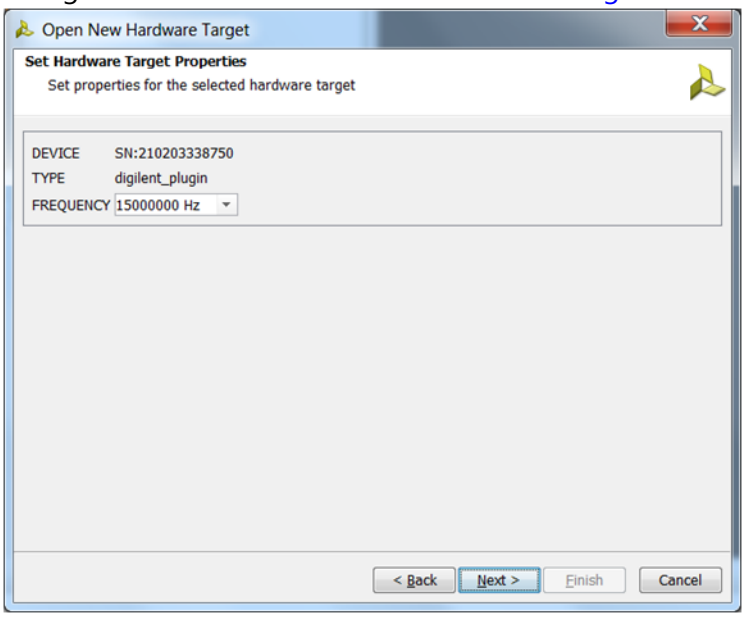

Figure 21: **CSE Hardware Target Parameter Settings**

7. Click **Finish** as shown in Figure 22.

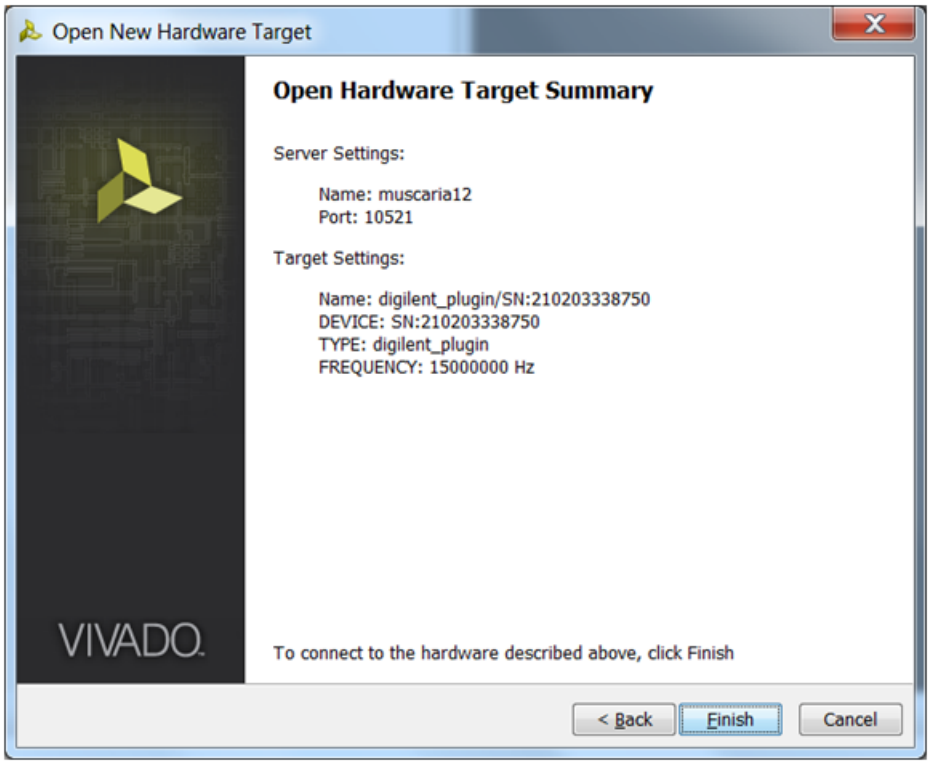

Figure 22: **Open Hardware Summary**

8. Wait for the connection to the hardware to complete. The dialog in Figure 23 appears while hardware is connecting.

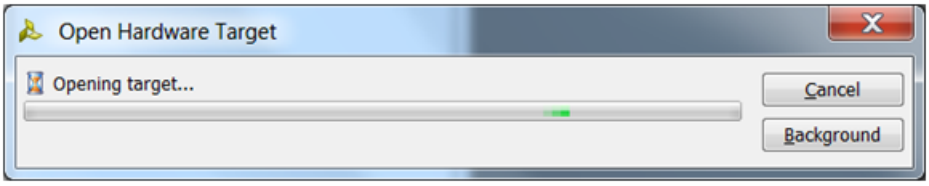

Figure 23: **Open Hardware Target**

9. Once the connection to the hardware target is made, the dialog shown in Figure 24 appears.

   Note: The **Hardware** tab in the **Debug** view shows the hardware target and **XC7K325T** device that was detected in the JTAG chain.

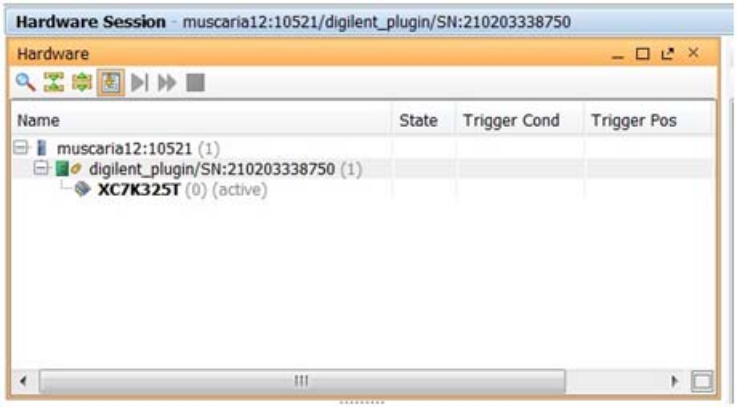

Figure 24: **Active Target Hardware**

10. Next, program the XC7K325T device using the `.BIT` bitstream file that was created previously by right-clicking on the **XC7K325T** device and selecting **Program Device** as shown in Figure 25.

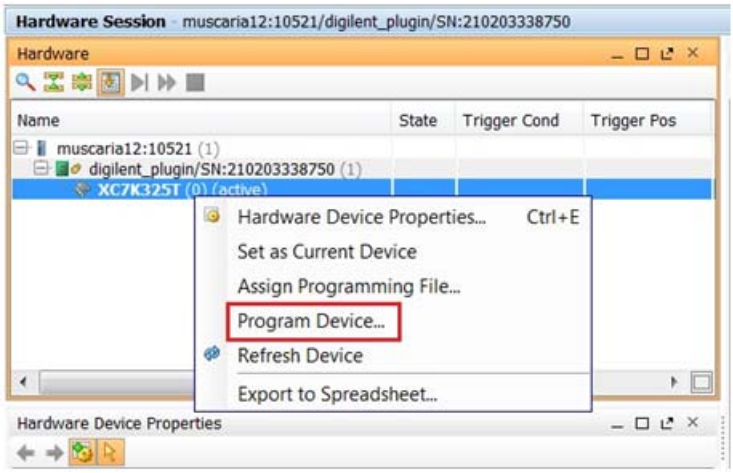

Figure 25: **Program Active Target Hardware**

11. Verify that the `.BIT` file is correct and click the **Program** button to program the device as shown in Figure 26.

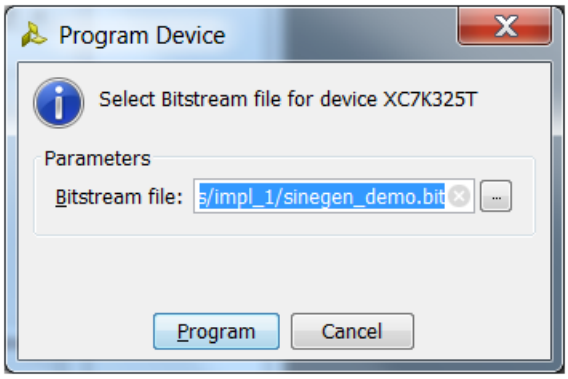

Figure 26: **Select Bitstream file to Download**

Note: Wait for the program device operation to complete. This may take few minutes.

12. Ensure that an ILA core was detected in the **Hardware** panel of the **Debug** view.

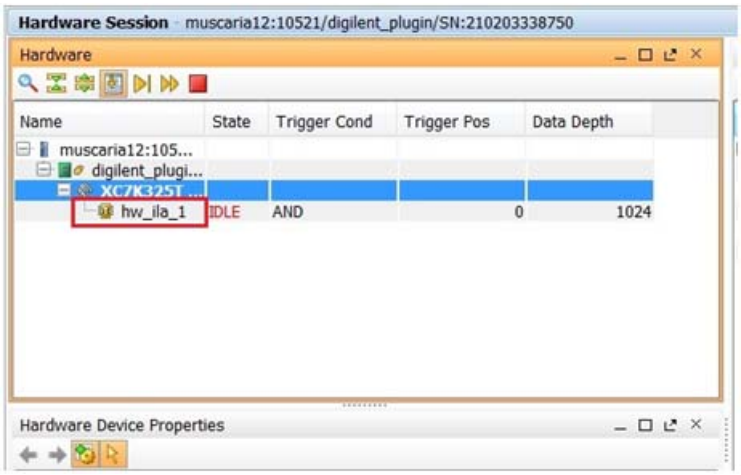

Figure 27: **ILA Core Detection**

13. Next, select the **ILA Probes** tab as shown in Figure 28 and ensure that all of the debug nets added from previous steps are accounted for.

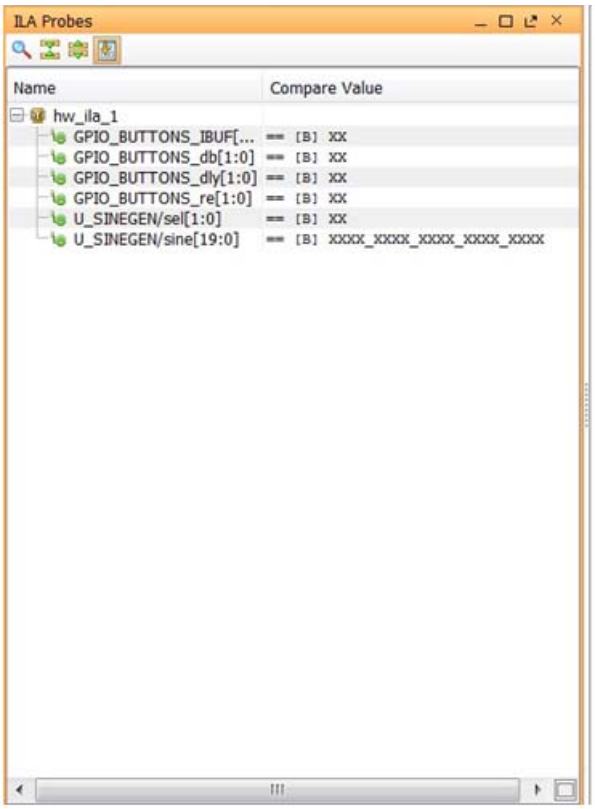

Figure 28: **List of Debug Nets under EDA_PROBES**

Verifying Sine Wave Activity

1. Click the **Run Trigger Immediate** button to trigger and capture data immediately as shown in Figure 29.

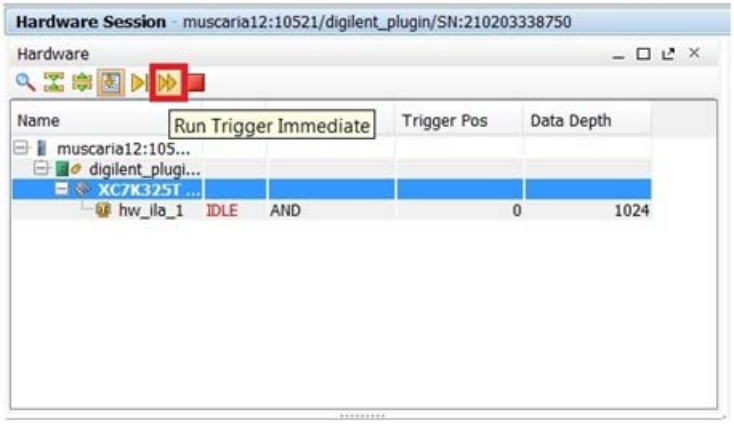

Figure 29: **Run Trigger Immediate Button**

2. In the **Waveform** window, verify that there is activity on the 20-bit sine signal as shown in Figure 30.

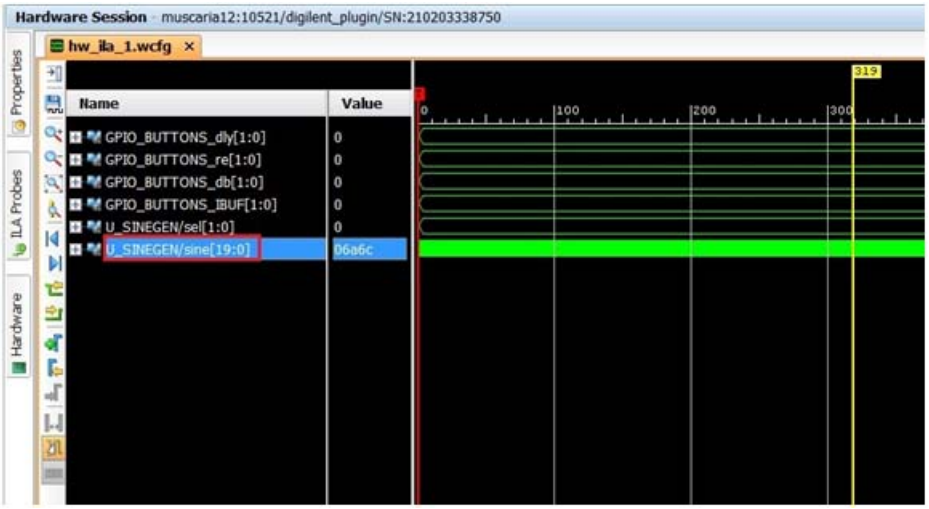

Figure 30: **Output Sine Wave Displayed in Digital Format**

Displaying the Sine Wave

1. Right-click `U_SINEGEN/sine[19:0]` signals, and select **Waveform Style >Analog** as shown in Figure 31.

Note: Notice that the waveform does not look like a sine wave. This is because you must change the radix setting from Hex to Signed Decimal, as described in the following subsection.

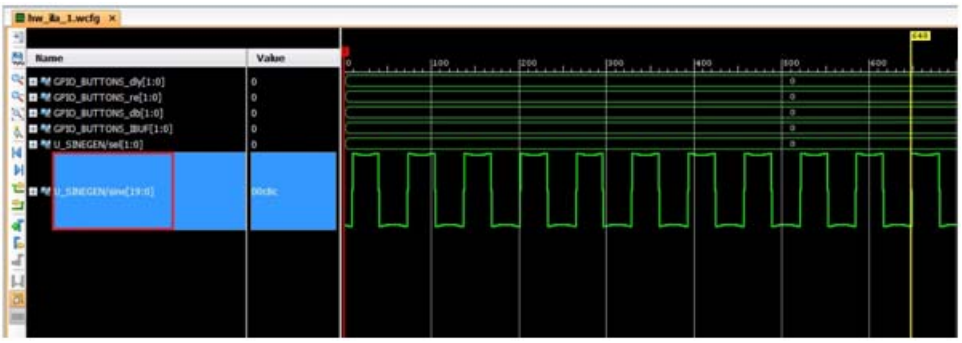

Figure 31: **Output Sine Wave Displayed in Analog Format, High Frequency - 1**

2. Right-click on `U_SINEGEN/sine[19:0]` signals, and select **Radix** > **Signed Decimal**. You should now be able to see the high frequency sine wave as shown in Figure 32 instead of the square wave.

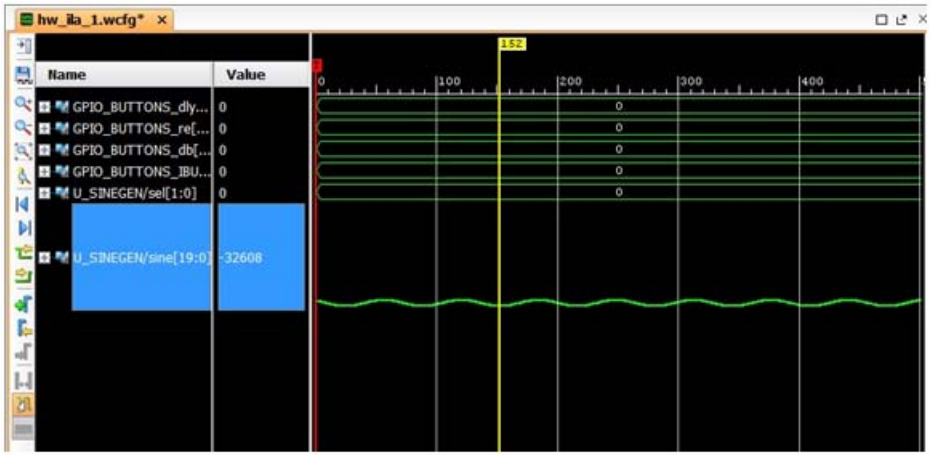

Figure 32: **Output Sine Wave Displayed in Analog Format, High Frequency - 2**

Correcting Display of the Sine Wave

To view the mid, and low frequency output sine waves, perform the following steps.

1. Cycle the sine wave sequential circuit by pressing the **GPIO_SW_E** push button as shown in Figure 33.

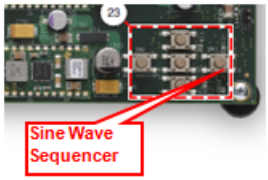

Figure 33: **Sine Wave Sequencer Push Button**

2. Click **Run Trigger Immediately** again to see the new sine selected sine wave. You should see the mid frequency as shown in Figure 34. Notice that the sel signal also changed from 0 to 1 as expected.

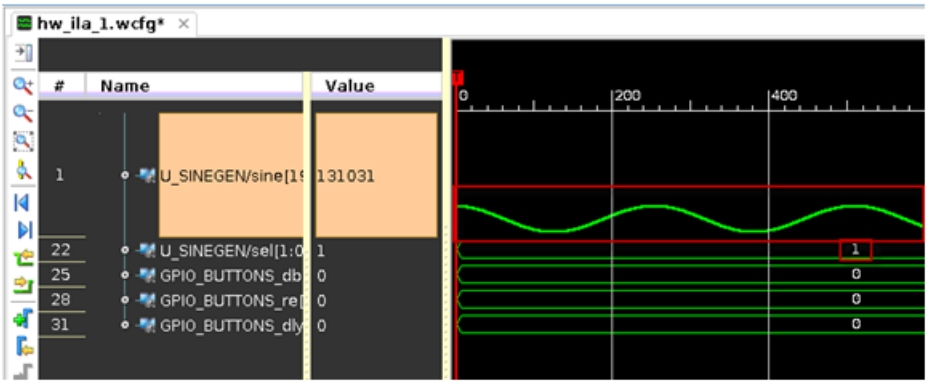

Figure 34: **Output Sine Wave Displayed in Analog Format – Mid Frequency**

3. Repeat step 1 and 2 from above to view other sine wave outputs

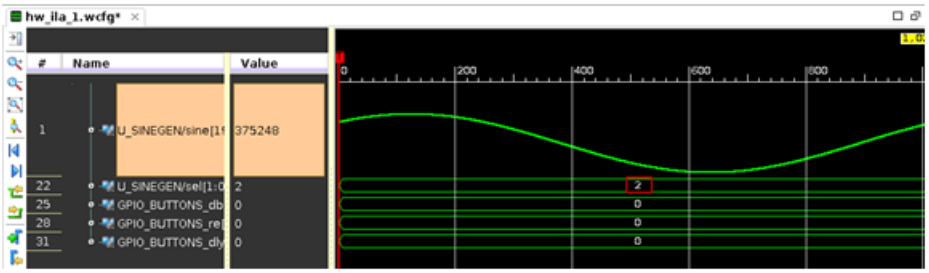

Figure 35: **Output Sine Wave Displayed in Analog Format – Low Frequency**

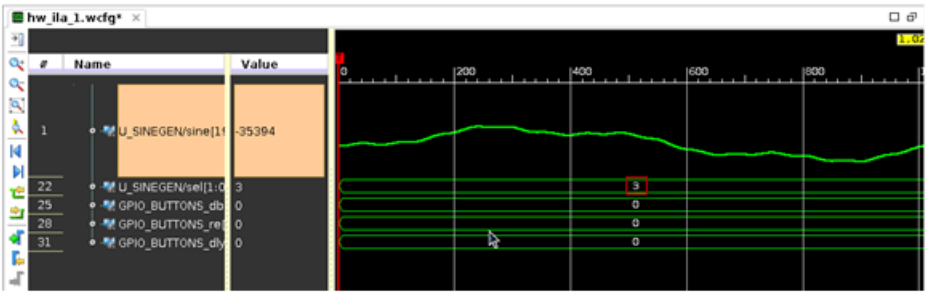

Figure 36: **Output Sine Wave Displayed in Analog Format – Mixed Frequency**

Note: As you sequence through the sine wave selections, you may notice that the LEDs do not light up in the expected order. You will debug this in the next section of this tutorial. For now, you will verify, for each LED selection, that the correct sine wave is displayed. Also note that the signals in the waveform window have been re-arranged in Figure 34, Figure 35, and Figure 36.

Debugging the Sine Wave Sequencer State Machine (Optional)

As you were correcting the sine wave display, the LEDs might not have lit up in sequence as you pressed the Sine Wave Sequencer button. With each push of the button, there should be a single, cycle-wide pulse on the `GPIO_BUTTONS_re[1]` signal. If there is more than one, the behavior of the LEDs becomes irregular. In this section of the tutorial, you will use Vivado logic analyzer to probe the sine wave sequencer state machine, and to view and repair the root cause of the problem.

Before starting the actual debug process, it is important to understand more about the sine wave sequencer state machine.

Sine Wave Sequencer State Machine Overview

The sine wave sequencer state machine selects one of the four sine waves to be driven onto the sine signal at the top-level of the design. The state machine has one input and one output. Figure 37 shows the schematic elements of the state machine. Refer to this diagram as you read the following description and as you perform the steps to view and repair the state machine glitch.

- The input is a scalar signal called button. When the button input equals '1', the state machine advances from one state to the next.
- The output is a 2-bit signal vector called Y, and it indicates which of the four sine wave generators is selected.

The input signal button connects to the top-level signal GPIO_BUTTONS_re[1], which is a low-to-high transition indicator on the Sine Wave Sequencer button (shown in Figure 1, page 5). The output signal Y connects to the top-level signal, sineSel, which selects the sine wave.

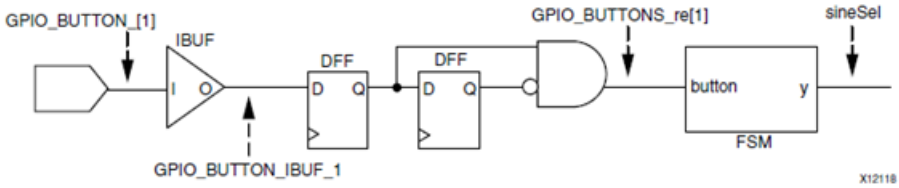

Figure 37: **Sine Wave Sequencer Button Schematic**

Viewing the State Machine Glitch

Viewing the Button Input to the Design

You cannot troubleshoot the issue you identified above by connecting a debug probe to the GPIO_BUTTON[1] input signal itself. The GPIO_BUTTON[1] input signal is a PAD signal that is not directly accessible from the FPGA fabric. Instead, you must trigger on

low-to-high transitions (rising edges) on the GPIO_BUTTON_IBUF signal (shown in Figure 37), which is connected to the output of the input buffer of the GPIO_BUTTON[1] input signal.

As described earlier, the glitch reveals itself as multiple low-to-high transitions on the GPIO_BUTTONS_1_IBUF signal, but it occurs intermittently. Because it could take several button presses to detect it, you will now set up the Vivado logic analyzer tool to **Repetitive Trigger Run Mode**. This setting makes it easier to repeat the button presses and look for the event in the **Waveform** viewer.

1. Set the trigger position of the ILA core to 512 (at the midway point of the 1024 sample-deep captured window shown in Figure 38).
2. Source the `rt.tcl` file in the Tcl command prompt. This Tcl commands perform the following tasks:
   - Sets the trigger position of the ILA core to 512 (at the midway point of the 1024 sample-deep captured window).
   - Looks for a rising edge transition on the GPIO_BUTTONS_IBUF transition.
   - Arms the trigger.
   - Waits for the trigger.
   - Uploads and displays waveforms.
3. On the KC705 board, press the Sine Wave Sequencer button until you see multiple transitions on the GPIO_BUTTONS_1_IBUF signal (this could take 10 or more tries). This is a visualization of the glitch that is occurring on the input. An example of the glitch is shown in Figure 38 and Figure 39.

Note: You might not observe signal glitches at exactly the same location as shown in the figure below.

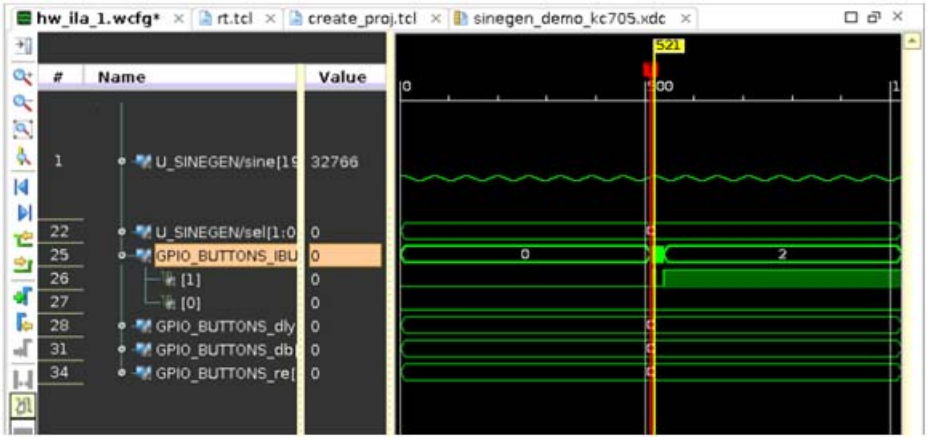

Figure 38: **GPIO_BUTTONS_BUF1 Signal Glitch**

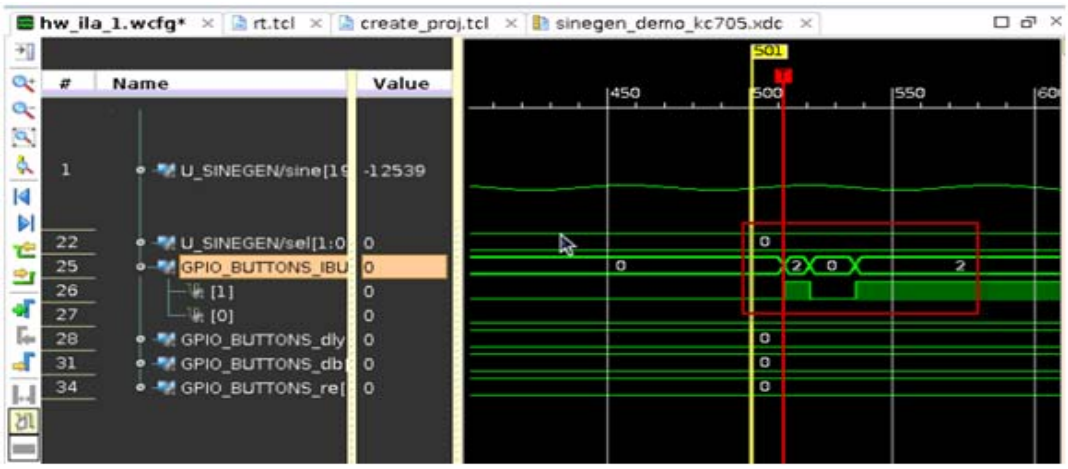

Figure 39: **GPIO Buttons_1_re Signal Glitch**

Fixing the Signal Glitch and Verifying the Correct State Machine Behavior

The multiple transition glitch or "bounce" occurs because the mechanical button is making and breaking electrical contact just as you press it. To eliminate this signal bounce, a "debouncer" circuit is required.

1. Enable the debouncer circuit by setting DIP switch position on the KC705 board (labeled Debounce Enable in Figure 1, page 5) to the **ON** or **UP** position.
2. Repeat step 2, page 35, and step 3, page 35, to:
   - Ensure that you no longer see multiple transitions on the GPIO_BUTTON_re[1] signal on a single press of the Sine Wave Sequencer button.
   - Verify that the state machine is working correctly by ensuring that the sineSel signal transitions from 00 to 01 to 10 to 11 and back to 00 with each successive button press.

Appendix A

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

See the Integrated Logic Analyzer (ILA) 2.0 product page for details on the latest information on *LogiCORE IP ChipScope Pro Integrated Logic Analyzer (ILA) (v2)*: http://www.xilinx.com/products/intellectual-property/ila.htm.

References

These documents provide supplemental material useful with this guide:

Vivado™ Design Suite 2012.3 Documentation (http://www.xilinx.com/cgi-bin/docs/rdoc?v=2012.3;t=vivado+docs)

ISE Design Suite 14.3 Documentation (http://www.xilinx.com/cgi-bin/docs/rdoc?v=14.3;t=ise+docs)

LogiCORE IP ChipScope Pro Integrated Logic Analyzer (ILA) (v2) Datasheet (DS875) (*http://www.xilinx.com/cgi-bin/docs/ndoc?l=en;t=data+sheet;d=ds875.pdf*).